Analysis of Runtime of Optimization Algorithms for Noisy Functions over Discrete Codomains

Youhei Akimoto[a], Sandra Astete-Morales[b], Olivier Teytaud[c]

[a]*Institute of Engineering, Shinshu University, Wakasato 4-17-1, Nagano, Japan*
[b]*TAO (INRIA) LRI, Bat 650 Ada Lovelace, Université Paris Sud, 91405, Orsay, France*
[c]*TAO (INRIA) LRI, Bat 650 Ada Lovelace, Université Paris Sud, 91405, Orsay, France*

Abstract

We consider in this work the application of optimization algorithms to problems over discrete codomains corrupted by additive unbiased noise.

We propose a modification of the algorithms by repeating the fitness evaluation of the noisy function sufficiently so that, with a fix probability, the function evaluation on the noisy case is identical to the true value.

If the runtime of the algorithms on the noise-free case is known, the number of resampling is chosen accordingly. If not, the number of resampling is chosen regarding to the number of fitness evaluations, in an anytime manner.

We conclude that if the additive noise is Gaussian, then the runtime on the noisy case, for an adapted algorithm using resamplings, is similar to the runtime on the noise-free case: we incur only an extra logarithmic factor. If the noise is non-Gaussian but with finite variance, then the total runtime of the noisy case is quadratic in function of the runtime on the noise-free case.

Keywords: Discrete Optimization, Additive noise, Runtime analysis

1. Introduction

1.1. State of the Art

In this work we focus on discrete optimization problems defined as minimization or maximization of functions from any domain to a discrete

Email addresses: y_akimoto@shinshu-u.ac.jp (Youhei Akimoto), sandra-cecilia.astete-morales@inria.fr (Sandra Astete-Morales), olivier.teytaud@inria.fr (Olivier Teytaud)

codomain. Since the feasible search space can be very large and the problem might be *hard* (from a computational complexity point of view), there has been a big development of *search heuristics* that find approximate solutions to the problems.

A strong motivation to develop this work is the immediate application of the results here presented to such heuristics. In particular, Evolutionary Algorithms (EAs) suit perfectly the framework where this paper is developed. EAs are important tools and have been successfully applied to discrete optimization problems [15, 17, 7, 25]. They are population-based black-box algorithms. Starting with an initial population of feasible search points (or candidates), the EAs *evolves* this population by selecting the *best* points of each generation. The selection is made using the *fitness* of the individuals which is not known explicitly thanks to the "black-box" setting. This is, no internal property of the *fitness function* is available during the optimization process; we have access to it only through its value for each individual. The *evolution process* on an EA is represented by several stages. The stages can be divided in : *Evaluation*, *Selection* and *Variation*. Even though these processes are easily described, the behavior of the EAs is hard to analyze rigorously.

For the theoretical analysis of EAs in discrete domains, authors usually consider some classical simple EAs such as the Randomized Local Search (RLS) method and the $(1+1)$-EA. For the objective functions, the OneMax function $OneMax_N : \{0,1\}^N \to \mathbb{N}, x \mapsto \sum_{i=1}^{N} x_i$ and the LeadingOnes function $LO_N : \{0,1\}^N \to \mathbb{N}, x \mapsto \sum_{i=1}^{N} \prod_{j=1}^{i} x_j$ have been analysed in many articles. We refer to [18] for the mathematical analysis of the runtime of these algorithms using drift analysis. Roughly speaking, the runtime of RLS or $(1+1)$-EA on $OneMax_N$ is $\Theta(N \ln(N))$ and the runtime of RLS or $(1+1)$-EA on LO_N is $\Theta(N^2)$. These bounds hold (in particular, but not only) with probability $1-\delta$ for a fixed $\delta > 0$. The results of this paper will provide as a consequence the runtime analysis of noisy counterparts of the previous results.

Evolutionary Algorithms exhibit robustness in the presence of noise. It has been shown that they are naturally robust in front of *actuator* [1] noise (see [14, 6]). While other studies refer to noisy fitness values, with noise models such as *additive* or *multiplicative* noise, as in [13]. In the work presented

[1] We have access to noisy search points instead of the real search points

here, we focus on the study of noisy functions such that the fitness values are perturbed by additive noise with constant variance all over the domain.

It is known that EAs, even in the discrete case, do not solve noisy objective functions without an appropriate modification: A. Prugel-Bennett, J. Rowe and J. Shapiro [20] have proved that with high probability, the RLS algorithm needs $\exp(cN)$ evaluations, for some $c > 0$ before finding the optimum of the $OneMax_N$ function with additive Gaussian noise with variance $\sigma^2 = 1$.

An alternative is to modify the algorithms for improving the manipulation of noisy fitness values. The most popular modifications consist in either resampling (aka averaging) the objective function [1], or the use of surrogate models to reduce the noise [4, 16, 3, 19]. With a surrogate model, an approximation of the objective function is learnt from the previous evaluations, and it is used to estimate the noise-free objective function values. That is, the surrogate models are supposed to reduce the impact of noise.

In resampling, instead of evaluating the fitness of a search point only once, as in a noise-free algorithm, the algorithm evaluates the fitness k times, obtaining k realizations of the noisy objective function on the specific search point. With those k *resamplings* of the noisy objective function on the search point[2], the algorithms averages them and uses this average as an approximate fitness value of the point. There are several variants for choosing k, such as: taking a fixed k, or incrementing k as the iteration does, or adapting k during the optimization. The impact of resampling on the convergence rate has been empirically or theoretically investigated in the references [23, 1, 12, 22, 21]. We here focus on the adaptation of resampling works from continuous codomains [2] to discrete ones and we cover a broad class of optimizers stated in the next subsection.

1.2. Framework

In this section we define the notations for the optimization algorithms that will be analyzed. Note that most EAs can be described as the algorithms defined in the following, but any optimization algorithm is potentially modifiable and will attain the same consequences.

First, we define the class of optimization algorithms, denoted by Opt, to solve a class G of noiseless and discrete objective functions $g : D \to \mathbb{Z}$, where

[2]It is assumed the noise observation is mutually independent noise realizations.

D is an arbitrary domain and the codomain $\mathbb{Z}$ is the set of integers. Then, we define a modification of *Opt* with k-times revaluation of the fitness, denoted by *k-Opt*, for a class of noisy fitness functions whose expectation is $g \in G$.

1.2.1. Formalization of optimization algorithms

We consider, very generally, that an optimization algorithm *Opt* at iteration n computes the next search point based on the previous iterations of the algorithm, as follows:

$$x_{n+1} = Opt(x_1, \ldots, x_n, y_1, \ldots, y_n) \ , \tag{1}$$

where $(x_i)_{i=1}^n$ is the sequence of search points in D and each $y_i = g(x_i)$, $g : D \to \mathbb{Z}$ is the objective function that belongs to some family G of objective functions. More formally, the optimization algorithm is defined by the map $Opt : \cup_{n \in \mathbb{N} \cup \{0\}} D^n \times \mathbb{Z}^n \to D$, where $Opt(\emptyset)$ defines the initial search point. Each iteration it generates a search point by *Opt* and evaluates the fitness g.

1.2.2. Formalization of resampling-based optimization algorithms

Now, we define *k-Opt*, the resampling-based version of an algorithm *Opt*, using a sequence k for choosing the number of revaluations.

In the noisy case, each evaluation of the fitness depends on the search point x_n and also on a random variable, namely ω: the objective function is no longer deterministic, but aleatory and denoted by $g(x, \omega)$. Then, the algorithm computes for each search point x_n an average fitness over k_n evaluations. We will denote this average

$$\hat{y}_n := \frac{1}{k_n} \sum_{i=1}^{k_n} g(x_n, \omega_{n,i}) \ , \tag{2}$$

where $\omega_{n,i}$ are independent copies[3] of a random variable ω. With these considerations, *k-Opt* depends on the sequence of search points and the sequence of averaged fitness values as follows:

$$\begin{aligned} x_{n+1} &= k\text{-}Opt(x_1, \ldots, x_n, \hat{y}_1, \ldots, \hat{y}_n) \\ &:= Opt(x_1, \ldots, x_n, R(\hat{y}_1), \ldots, R(\hat{y}_n)) \end{aligned} \tag{3}$$

[3]For example, $g(x, \omega) = g(x) + \omega$ with ω a standard Gaussian random variable, in the case of the classical Gaussian additive noise.

where $R(\cdot)$ represents a *round function* (more precisely, for any x, $R(x) = \lfloor x + \frac{1}{2} \rfloor$), and $k = (k_i)_{i \in \mathbb{N}}$ represents the sequence of the number of revaluations for the search points. Hence, Eq. (3) defines k-*Opt*, which is the "resampling counterpart" of *Opt*. In other words, k-*Opt* works exactly as *Opt* except that it modifies the way to obtain the objective function value of a search point: k-*Opt* averages several realizations of the noisy fitness functions and it uses a round function to assign an integer fitness value to each search point and continue with the optimization process as in *Opt*.

1.3. Outline of the paper

In this paper, we provide an upper bound on the runtime or many other performance measure of k-*Opt* on a class of noisy problems, with or without knowing in advance the runtime of *Opt* on the noisefree counterparts of these problems.

Our results are divided into two parts (Sections 2 and 3). In Section 2, we assume that we know the runtime of the algorithm *Opt* on a class of noiseless problems and we pre-tune the *fixed* number of resampling k. This is the case of a preknown runtime. We derive the runtime of k-*Opt* in the presence of two different types of noise: Gaussian (Section 2.1) and any noise with finite variance (Section 2.2).

In Section 3, dedicated to the anytime analysis, we remove the assumption that the runtime of *Opt* is pre-known. We design the sequences of the numbers k_n of resampling at each iteration n for the Gaussian noise and any noise with finite variance. Then we show that with a high probability the sequence of the solutions and their averaged and rounded function values generated by k-*Opt* on a class of noisy problems satisfies any quality measure or any formula that is satisfied by those generated by *Opt* on the class of noiseless counterparts. It implies that if we know the runtime, say N, of *Opt*, we can derive the runtime of k-*Opt* on the noisy counterpart as $\sum_{n=1}^{N} k_n$.

In Section 4 we discuss the tightness of the results presented, state some straightforward consequences over some classic evolutionary algorithms and present some comments on the definition of the runtime. In Section 5 we conclude this work.

2. Analysis in the pre-known runtime case

We here consider the case in which the runtime is pre-known, i.e. this is not the anytime case. Let *Opt* be an optimization algorithm as de-

fined in Equation (1), and G be a family of fitness function $g : D \to \mathbb{Z}$. Assume that each fitness or objective function $g \in G$ has its optimum $x^* := \arg\max_{x \in D} g(x)$. The definition of *runtime* is stated as follows:

Definition 2.1 (Runtime $r(\delta)$ of an algorithm *Opt* to solve G)**.** The runtime $r(\delta) = r(G, \delta)$ of an algorithm *Opt* to solve G is the number of fitness evaluations needed before it evaluates the optimum of any function $g \in G$, with probability at least $1 - \delta$.

In this section, we derive the upper bound of the runtime of k-*Opt* defined in (3), the *Opt* with k resampling, to solve the family of noisy functions whose expectation is an element of G. We assume that we know the runtime $r(\delta)$ of the underlying algorithm *Opt* to solve G and the variance of the noise, σ^2. The case that we do not know the runtime of *Opt* in advance is studied in the next section.

Note that k in (3) is a sequence and each element k_n of k depends on each iteration $n \in \mathbb{N}$ of the algorithm. In this analysis we define k depending only on the pre-known runtime $r(\delta)$ of *Opt* to solve G and it will remain the same for each iteration. With a slight abuse of notation, we denote $k_n = k \in \mathbb{N}$ for any $n = \mathbb{N}$.

The idea behind our choice of k is to have enough revaluations of the noisy fitness value of each point to maintain the behavior of the algorithm *Opt* in the noise-free case. We choose k as a function of the runtime $r(\delta)$ in the noise-free case. This is why we consider that this is not anytime; we assume that we know *a priori* the budget used by the algorithm *Opt* to solve G.

2.1. Runtime analysis with resampling - Gaussian noise

Let $G + \sigma\mathcal{N} := \{g + \sigma\mathcal{N} | g \in G, \mathcal{N}$ is a standard Gaussian noise$\}$ be the family of fitness functions with additive Gaussian noise with variance σ^2. We study the runtime of the algorithm k-*Opt* to solve $G + \sigma\mathcal{N}$.

Theorem 2.2. *Assume that Opt solves G with runtime $r(\delta)$. Define*

$$k_{gauss} = \max\left(1, \left\lceil 32\sigma^2 \left[\ln(2) - \ln\left(1 - (1-\delta)^{1/r(\delta)}\right)\right]\right\rceil\right). \tag{4}$$

Then, k_{gauss}-Opt solves $G+\sigma\mathcal{N}$ with probability at least $(1-\delta)^2$ and a runtime

$$O\left(r(\delta)\max\left(1, \sigma^2 \ln\left(\frac{r(\delta)}{\delta}\right)\right)\right).$$

Proof. In the following we consider that the *noisy fitness value* of an individual x is the average over k evaluations of the noisy fitness of the individual (see Eq. 2). Accordingly, *Opt* has access to the real fitness value of x while *k-Opt* has access to the noisy fitness value of x, as shown in Eq. 3. We compute now the probability p of the noisy fitness value of point x being separated from the noise-free fitness value by at least $1/4$. Computing this probability will allow us to know the opposite: the probability of real fitness value and noisy fitness value to be *sufficiently close*. In this context sufficiently close means that *k-Opt* assigns the same fitness value as the real one thanks to the round function R.

Since the noise is Gaussian with variance σ^2 and observed fitness values are mutually independent, the difference between the averaged noisy fitness value $\hat{y}$ and the noise-free fitness y at any point x follows a Gaussian distribution with variance σ^2/k. Thus,

$$p = \Pr\left(|\mathcal{N}| > \frac{\sqrt{k}/4}{\sigma}\right) = 2\,\mathrm{erfc}\left(\frac{\sqrt{k}/4}{\sqrt{2\sigma^2}}\right) \leq 2\exp\left(-\frac{k/4^2}{2\sigma^2}\right) \quad . \tag{5}$$

Here erfc denotes the complementary error function and we used inequality $\mathrm{erfc}(x) \leq \exp(-x^2)$ [9].

Then, $(1-p)^{r(\delta)}$ represents the probability that the distance between the fitness value and the noisy fitness value is bounded by $1/4$ for all samples generated by *k-Opt* in run length $r(\delta)$.

Using k_{gauss} defined in (4), and the bound deduced in (5), we obtain

$$\delta \geq 1 - (1-p)^{r(\delta)}.$$

This is exactly the bound we were looking for: the probability that we get a misranking, at least once in run length $r(\delta)$, is upper bounded by δ, the probability of failure of the optimizer *Opt* in the noise-free case (see definition 2.1).

The overall number of function evaluations is $r(\delta) \cdot k_{gauss}$. Using the general inequality $(1-x) \leq y(1-x^{1/y})$ for $0 \leq x \leq 1$ and $y \geq 1$, since $r(\delta)$ is assumed to be no less than 1 without loss of generality, we have from (4)

$$k_{gauss} \in O(\sigma^2 \ln(r(\delta)/\delta)) \quad .$$

Then, we finally obtain

$$r(\delta) \cdot k_{gauss} \in O\left(r(\delta)\max\left(1, \sigma^2 \ln\left(\frac{r(\delta)}{\delta}\right)\right)\right) \quad . \tag{6}$$

Equation (6) concludes the proof. □

We remark that Qian et al. [21] has investigated the impact of the resampling in the 1+1-EA on the expected first hitting time. They have concluded in Theorem 2 of [21] that the resampling is useless for $1+1$-EA optimizing the OneMax with Gaussian noise, i.e., the expected first hitting time will increase as k increases. Our result does not say anything about the runtime of *Opt* on a noisy problem, but states how many function evaluations we will use when *k-Opt* is employed to optimize a class of noisy problems compared to when *Opt* is used to solve the class of noiseless counterpart. Moreover, our result is not restricted to a specific $1+1$-EA.

We would like to also remark that the result obtained here and its proof idea is similar to Theorem 1 of [11], where the expected first hitting time of a simple algorithm for noisy bi-objective optimization problems is analyzed. However, we consider a wide class of algorithms, a large class of objective functions, and we distinguish noises with light and heavy tail. On the other hand, we consider a mono-objective case only.

2.2. Runtime analysis with resampling - heavy tail scenario

In the previous section, we have assumed that a Gaussian noise perturbs the objective functions. More general cases in which the distribution tail decreases quickly lead to similar bounds, but with worse performance of the modified algorithms. In this paper, we refer to the heavy tail case when we have no assumption on the noise distribution, except that the variance is finite.

The following theorem suggests that the heavy tail case is *harder*, in the sense that it increases the runtime to quadratic — however, we have no lower bound.

Theorem 2.3. *Assume that an optimizer Opt solves G_n with runtime $r(\delta)$. Define $k_{heavy} = \max(1, \lceil 16\sigma^2/(1-(1-\delta)^{r(\delta)})\rceil)$ in dimension N. Then k_{heavy}-Opt solves $G + \mathfrak{N}$, where $\mathfrak{N}$ is an arbitrary noise with mean 0 and variance σ^2, with probability at least $(1-\delta)^2$ and runtime*

$$O\left(r(\delta)\max\left(1, \sigma^2\frac{r(\delta)}{\delta}\right)\right). \tag{7}$$

Proof. The proof is essentially the same as the proof for Theorem 2.2, but Eq. (5) becomes less convenient due to the non-Gaussian nature of noise.

We get a bound by Chebyshev's inequality rather than with the Gaussian cumulative distribution function: for a random variable with expectation zero and variance σ^2, the probability of a deviation between the averaged fitness $\hat{y}$ over k resamplings and the noise-free fitness y by at least ϵ is bounded as

$$P(|y - \hat{y}| \geq \epsilon) \leq \sigma^2/(k\epsilon^2) \ . \tag{8}$$

The probability of a point having an average fitness deviating by at least $\epsilon = 1/4$ from its expectation is no greater than $16\sigma^2/k$. So, the probability of at least one such deviation over $r(\delta)$ fitness evaluations is upper bounded by

$$1 - \left(1 - \frac{16\sigma^2}{k}\right)^{r(\delta)} \ ,$$

which is no greater than δ if $k \geq 16\sigma^2/(1-(1-\delta)^{1/r(\delta)})$. Since k_{heavy} defined in the theorem statement satisfies this condition, k_{heavy}-*Opt* on the noisy fitness exactly simulates the behavior of *Opt* on its noise free counterpart with probability at least $1-\delta$. Since *Opt* solves G within $r(\delta)$ with probability at least $1-\delta$, k_{heavy}-*Opt* solves $G + \sigma\mathfrak{N}$ with probability at least $(1-\delta)^2$ and the number of function evaluation

$$r(\delta) \cdot k_{heavy} \leq r(\delta)\lceil 16\sigma^2 r(\delta)/\delta \rceil \ .$$

This concludes the proof. □

3. Extension: anytime analysis and arbitrary criteria

We generalize previous results in two directions:

- We now consider a setting in which we do not know in advance the number of evaluations that *Opt* requires to solve a family of functions G.
- We also generalize to arbitrary criteria, as explained by the general Q criterion below.

An important remark about the definition of *runtime* used in this work (see Definition 2.1), is the following: we have defined the runtime as *the first hitting time*. This means that the algorithm must sample the optimum at least once and it does not necessarily have to stop when it hits the optimum.

It would be possible to define as a criteria the first "finding" time: the number of fitness evaluations needed before an algorithm actually stops and outputs the optimum of the objective function (with probability $1-\delta$). Another possibility is to define the the first "stabilization" time: the number of fitness evaluations n_0 such that for all $n \geq n_0$ the current guess of the optimum is the best (with probability $1-\delta$).

Other optimization criteria distinguish the search points and the approximation of the optima proposed by the algorithm. After all, it is not necessarily a problem if very *bad* individuals are evaluated, as long as the algorithm uses such information for guessing where is the optimum. It is noteworthy that classical algorithms, in the continuous setting, do sample points with not so good fitness values, to obtain information on the shape of the objective function [10, 5].

A fortiori in the noisy optimization case, the differences between the definitions of the runtime of an algorithm can have big consequences when there is a comparison between the noise-free case and the noisy case. This because finding the optimum in a search space $\{0,1\}$ (in case of dimension $N=1$) is trivial after two fitness evaluations for the first hitting time, whereas in case of an additive noise with large variance, finding with high probability which of the two search points is the best, might take a lot of time. We cannot always recommend the best value found so far, as is usually done in the noise-free case. The results in Section 3 cover a wide range of criteria, including criteria depending on an infinite sequence of iterates (see Eq. 9).

3.1. Formalization of general criteria in the anytime setting

We consider a criterion or a general formula that measure the performance of the algorithm *Opt*. Consider $Q : (\cup_{n\in\mathbb{N}\cup\{\infty\}} D^n \times \mathbb{Z}^n) \to \{0,1\}$. If $(x_n)_{n\leq M}$ and $(y_n)_{n\leq M}$ be the sequence of the search points and their fitness values generated by the algorithm *Opt*, we measure the performance of the *Opt* at iteration M (possibly $M=\infty$, i.e. the success criterion can take into account infinite sequences) by $Q((x_n)_{n\leq M},(y_n)_{n\leq M})$. This very general formalism allows a wide range of criteria.

In the following, we assume that *Opt* satisfies a criterion $Q\left((x_n)_{n\leq M},(y_n)_{n\leq M}\right) = 1$ for any $M \in \mathbb{N}$ with probability at least $1-\delta$ for any objective function $g \in G$:

$$\forall g \in G, \forall M \in \mathbb{N}, P(Q((x_n)_{n\leq M},(y_n)_{n\leq M}) = 1) \geq 1-\delta. \tag{9}$$

For example, if we define $Q((x_n),(y_n)) = \mathbb{I}\{\exists n \leq \tau \text{ s.t. } y_n = \min_x f(x)\}$, inequality (9) indicates that the first hitting time of *Opt* with probability at least $1-\delta$ is upper bounded by τ.

3.2. Gaussian case

The following theorem shows that if the noise is Gaussian, choosing a specific sequence k we can guarantee that *k-Opt* satisfies $Q((x_n),(R(\hat{y}_n)))$ with probability at least $(1-\delta)^2$. Note that x_n and y_n are the point and its fitness after $K_n = \sum_{i=1}^{n} k_n$ function evaluations. We define $\kappa = \{1+K_n; n \in \mathbb{N}\}$.

Theorem 3.1 (Gaussian noise, anytime case)**.** *Assume that Opt, defined as in Eq.* (1)*, satisfies a formula $Q((x_n),(y_n))$ for any objective function $g \in G$ with probability at least $1-\delta$. Consider $G+\sigma\mathcal{N}$ the noisy counterpart of G. Then, k-Opt as defined in Eq.* (3)*, with the sequence $k = (k_n)_{n\geq 1}$ given for any $\beta > 1$ by*

$$k_n = \left\lceil 32\sigma^2 \ln\left(\frac{2(n+1)\ln(n+1)^\beta}{\delta}\left(\sum_{i=2}^{\infty}\frac{1}{i\ln(i)^\beta}\right)\right)\right\rceil \tag{10}$$

satisfies $Q((x_n)_{n\in\kappa},(R(\hat{y}_n)_{n\in\kappa}))$ for any function in $G+\sigma\mathcal{N}$ with probability at least $(1-\delta)^2$. Additionally, the total number K_n of fitness evaluations up to the n-th iteration is $O(n\ln(n))$.

Remark 3.2. We might rephrase the theorem with criteria Q involving the expected fitness values rather than observed fitness values (including the noise). In this case we would then consider $Q((x_n)_{n\in\kappa},(\mathbb{E}_\omega f(x_n,\omega))_{n\in\kappa})$.

Proof. First, note that $\sum_{i=2}^{n}\frac{1}{i\ln(i)^\beta}$ that appears in Eq. (10) is known as Bertrand series and it is convergent as $n\to\infty$ for any $\beta > 1$.

Define $e_n = P(|y_n - \hat{y}_n| \geq 1/4)$ and $f_n = P(\exists i \leq n; |y_n - \hat{y}_n| \geq 1/4)$. Applying Eq. (5), we get that $e_n \leq 2\exp(-k_n/32\sigma^2)$. If we choose k_n as defined in Eq. (10), we have

$$e_n \leq \frac{\delta}{\left((n+1)\ln(n+1)^\beta \sum_{i=2}^{\infty}\frac{1}{i\ln(i)^\beta}\right)}$$

Then,

$$f_n \leq \sum_{i=1}^{n} e_i \leq \sum_{i=1}^{\infty} e_i \leq \delta$$

This implies that, with probability $(1-\delta)$, the rounded fitness evaluations are exactly the real fitness evaluations, i.e. $y_n = R(\hat{y}_n)$ for all $n \geq 1$. Therefore, with probability at least $(1-\delta)^2$, we have $Q((x_n), (R(\hat{y}_n)))$. The total number of function evaluations is then $K_n = \sum_{i=1}^{n} k_i \in O(\sum_{i=1}^{n} \ln(i)) = O(n \ln(n))$. This completes the proof. □

3.3. Heavy tail case

The follow theorem shows the heavy tail counterpart of the previous theorem.

Theorem 3.3 (Heavy tail, anytime case)**.** *Assume that Opt as defined in Eq.* (1) *satisfies a formula $Q((x_n), (y_n))$ for any objective function f in some class G with probability at least $1-\delta$. Consider $G+\sigma\mathfrak{N}$ the noisy counterpart of G, $\mathfrak{N}$ any noise with mean* 0 *and variance* 1*. Then, k-Opt as defined in Eq.* (3) *with the sequence k given for any $\beta > 1$ by*

$$k_n = \left\lceil \frac{16\sigma^2(n+1)\ln(n+1)^\beta}{\delta} \sum_{i=2}^{\infty} \frac{1}{j \ln(j)^\beta} \right\rceil \quad . \tag{11}$$

satisfies $Q((x_n)_{n\in\kappa}, (R(\hat{y}_n))_{n\in\kappa})$ for any function in $G+\sigma\mathfrak{N}$ with probability at least $(1-\delta)^2$. The total number of fitness evaluations up to the n-th iteration is $K_n \in O(n^2 \ln(n)^\beta)$.

Proof. The proof is the same as for Theorem 3.1, except that we use Chebyshev's inequality (8) instead of Eq. (5). □

These theorems indicate that, without knowing the runtime in advance, *k-Opt* requires $O(n \ln(n))$ function evaluations to guarantee any formula $Q((x_n), R(\hat{y}_n))$ to hold if the noise is normally distributed. If the noise has a heavy tail distribution, the required number of function evaluations grows up to $O(n^2 \ln(n)^\beta)$ for some $\beta > 1$, which is squared comparing to the Gaussian case if $\beta = 2$.

4. Discussion

In this section we present discussions on some aspects of this work. Namely the *tightness* of the results and the application of the results on this paper over known algorithms. For the tightness we conclude that the result is *approximately tight* in the Gaussian case, but not always in the heavy-tail case.

4.1. Approximate tightness in the Gaussian case

We here show the tightness of the upper bound, obtained in the Gaussian case, up to logarithmic factors.

The runtime of k-Opt in the noisy case is not much more than the runtime of Opt in the noise-free case . Let $T(\delta, G, Opt)$ be the number of evaluations guaranteed by some algorithm Opt before hitting the optimum of any $g \in G$ with probability $1-\delta$. We define also $T(\delta, G, Opt^*) = \inf_{Opt} T(\delta, G, Opt)$, Opt^* represents the algorithms that attains the optimum (assuming that the optimum is reached). Analogous for the noisy environment, $T_{noise}(\delta, G, \mathcal{N}, k\text{-}Opt^*)$ is the number of evaluations guaranteed by k-Opt^* (Opt^* with k revaluations) before hitting the optimum of any $g \in G + \sigma\mathcal{N}$ with probability $(1-\delta)^2$ and where Opt^* is the algorithm that reaches the optimum in the noise-free case.

By Theorem 2.2 and using $(1-\delta)^2 \geq 1-2\delta$ we show that

$$T_{noise}(\delta, G, \mathcal{N}, k\text{-}Opt^*) = O(T(2\delta, G, Opt^*) \ln(T(2\delta, G, Opt^*))) \tag{12}$$

The noisy case is harder than the noise-free case. We have shown that the method with revaluations, namely k-Opt, does not need much more evaluations than Opt in the noise free case. For a rigorous approximate tightness result, we must also show that there is no algorithm which is faster, in the noisy case, than the optimal algorithm in the noise-free case. This is proved by considering, for the family G of problems, a confidence $1-\delta$, and a Gaussian noise $\mathcal{N}$:

- An algorithm A, with optimal running time $T_A = T(\delta, G, A)$ for solving the task in the noise-free case with probability $1-\delta$ (here, "time" refers to the number of function evaluations) [4].
- The algorithm B, solving the noisy counterpart of the task with optimal running time $T_B = T_{noise}(\delta, G, \mathcal{N}, B)$ in the noisy case (standard Gaussian noise) with probability $1-\delta$. We do not necessarily consider here algorithms based on resamplings.
- Let us assume that $T_B < T_A$. This means that the noisy case is easier (in this precise sense that $T_B < T_A$).

[4] We point out that the proof above is based on the assumption that the optimal running time exists (i.e. the inf is reached). Otherwise we should just adapt the proof by considering algorithms optimal within some arbitrarily small $\epsilon > 0$.

- Then, the algorithm B, applied to the noise-free case and artificially adding a standard Gaussian noise, would perform the same task as A in time $T_B < T_A$, which is a contradiction.

This shows that the noisy case cannot be easier than the noise-free case, and that therefore the result is tight, up to the additional risk (confidence $(1-\delta)^2$ instead of $(1-\delta)$) and the logarithmic term in the runtime.

Conclusion: approximate tightness. Eq. 12 and this remark, together, show the tightness up to a logarithmic factor on the computation time and up to the change in δ.

4.2. Heavy tail case: no tightness

In the heavy tail case, we obtain Eq. 13 by definition and from Theorem 2.3

$$T(\delta, G, Opt^*) \leq T_{noise}(\delta, G, \mathfrak{N}, k\text{-}Opt^*) = O(T(2\delta, G, Opt^*)^2) \tag{13}$$

Constants in the $O(\cdot)$ depend solely on δ.

This is a non-negligible increase in the runtime: we jump to the square, compared to the noise-free case. A natural question would be: Is it possible to improve this result? There are problems and criteria for which the bound is not tight. For example, for needle-in-the-haystack (the fitness function equal to 0 in every search point, except at the optimum with value 1), the runtime is $\Theta(2^N)$ both in the noise-free and noisy cases if we use the first hitting time as a criterion[5]. This is however due to the first hitting time criterion; with such a criterion, the optimal algorithm for needle-in-the-haystack, both in the noisy and noise-free setting, is just to try each possible search point once. The tightness, without Gaussian assumption, with other (better) criteria is an open problem.

Remark 4.1 (Tightness for other criteria than the first hitting time ?)**.** As mentioned previously, even if the results on this paper are obtained for the *first hitting time* criteria, the same transformation can be applied to other criteria, as shown in Section 3. Meanwhile, the counter-example of the needle-in-the-haystack, showing that the complexity is not necessarily increased when switching to the noisy case, does not immediately work for other criteria. We do not know if the quadratic bound of this paper is approximately optimal for some criteria.

[5]Easily proved by trying each search point once

4.3. Consequences

Here we state some straightforward consequences on classical algorithms and problems. The results on section 2 are applied directly to algorithms with known runtime on the noise-free case. This means, having the information on the runtime, we modify the algorithm as defined in section 1.2.2, with *ad hoc* choice of k in each case.

In Table 1 we present the results. The first columns refers to the name of the problem. The second is the algorithm used to solve it. Third and fourth column are the runtime on the noise-free case and their reference respectively. The fifth and sixth columns refer to the results of the present work: using *k-Opt* as the revaluation counterpart of *Opt* applied in the case of Gaussian noise and any noise with finite variance (or heavy tailed). Note that we neglect the precision as a function of δ and the exact computation of k, to focus on the runtime dependency on the dimension N.

Table 1: Summary of consequences over known algorithms with their runtime

Problem	*Opt*	Runtime	Ref.	**Gaussian noise**	**Finite variance**
OneMax	$(1+1)$ EA	$O(N \log N)$	[17]	$O(N(\log N)^2)$	$O(N^2(\log N)^2)$
LO	Alg. in [8]	$O(N^2)$	[8]	$O(N^2 \log N)$	$O(N^4)$
Max Clique	$(1+1)$ EA	$O(N^5)$	[25]	$O(N^5 \log N)$	$O(N^{10})$
Sorting	$(1+1)$ EA	$O(N^2 \log N)$	[24]	$O(N^2(\log N)^2)$	$O(N^4(\log N)^2)$

5. Conclusion

In this work we have shown that slights modifications of algorithms prepared to solve noise-free instances can lead to adapted algorithms that can solve the noisy counterpart of the instances. The noise is modeled as additive and in two configurations: Gaussian noise and heavy tailed noise. Heavy tailed in this context refers to any noise with finite variance.

However, the main focus of this paper was to find conclusions over the runtime of the algorithms. Firstly, we consider we know in advance the runtime of an algorithm over an instance. With this information, we can modify the original algorithm to obtain one that can handle the noisy instance, adding a constant number of revaluation in each iteration. This number depends on the runtime of the algorithm on the noise-free case. We conclude that solving a problem with Gaussian additive noise, is *similar* to the runtime in the noise-free case: just an additional logarithmic factor. Meanwhile, for

the heavy tail additive noise, the runtime is quadratic on the runtime of the algorithm on the noise-free case.

Furthermore, these bounds hold in the anytime case: an arbitrary criteria depending on the sequence of search points and which holds in the noise-free case, also holds in the noisy case, up to a logarithmic factor (in the Gaussian case) or a squared runtime (heavy tail case).

The bound for the Gaussian noise case is *approximately tight* in the sense that if *Opt* is optimal in the noise-free case, *k-Opt* is nearly tight (up to logarithmic factors) in the noisy case, in terms of dependency in the dimension N. The bound for the heavy tail case is not tight because some problems with large complexity have the same dependency in N in the noisy case and in the noise-free case.

Further work. A natural further work is continuous codomains, with limited requested precision - i.e. when the codomain is $\mathbb{R}$, but we "only" have to find x such that $\mathbb{E}_\omega f(x,\omega) \geq \sup_{x'} \mathbb{E}_\omega f(x',\omega) - \epsilon$.

Acknowledgements

This work benefited from discussions between Y. Akimoto, S. Astete-Morales, A. Prugel-Bennett, J. Rowe, J. Shapiro, O. Teytaud, originating at Dagstuhl seminar 13271.

[1] Arnold, D., Beyer, H.G., 2006. A general noise model and its effects on evolution strategy performance. Evolutionary Computation, IEEE Transactions on 10, 380–391. doi:10.1109/TEVC.2005.859467.

[2] Astete-Morales, S., Liu, J., Teytaud, O., 2014. Log-log convergence for noisy optimization, in: Legrand, P., Corsini, M.M., Hao, J.K., Monmarché, N., Lutton, E., Schoenauer, M. (Eds.), Artificial Evolution. Springer International Publishing. volume 8752 of *Lecture Notes in Computer Science*, pp. 16–28. doi:10.1007/978-3-319-11683-9_2.

[3] Booker, A.J., Dennis, J.E., Jr., Frank, P.D., Serafini, D.B., Torczon, V., 1998. Optimization using surrogate objectives on a helicopter test example, in: RICE UNIVERSITY, SIAM. pp. 49–58.

[4] Caballero, J.A., Grossmann, I.E., 2008. An algorithm for the use of surrogate models in modular flowsheet optimization. AIChE Journal 54, 2633–2650. doi:10.1002/aic.11579.

[5] Coulom, R., 2012. Clop: Confident local optimization for noisy black-box parameter tuning, in: Advances in Computer Games. Springer Berlin Heidelberg. volume 7168 of *Lecture Notes in Computer Science*, pp. 146–157. doi:10.1007/978-3-642-31866-5_13.

[6] Droste, S., 2004. Analysis of the (1 + 1) ea for a noisy onemax, in: Deb, K. (Ed.), Genetic and Evolutionary Computation – GECCO 2004. Springer Berlin Heidelberg. volume 3102 of *Lecture Notes in Computer Science*, pp. 1088–1099. doi:10.1007/978-3-540-24854-5_107.

[7] Droste, S., Jansen, T., Wegener, I., 2002. On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science 276, 51 – 81. doi:http://dx.doi.org/10.1016/S0304-3975(01)00182-7.

[8] Droste, S., Jansen, T., Wegener, I., 2006. Upper and lower bounds for randomized search heuristics in black-box optimization. Theor. Comp. Sys. 39, 525–544. doi:10.1007/s00224-004-1177-z.

[9] Ermolova, N., Haggman, S.G., 2004. Simplified bounds for the complementary error function; application to the performance evaluation of signal-processing systems, in: Proceedings of the 12th European Signal Processing Conference, pp. 1087–1090.

[10] Fabian, V., 1967. Stochastic Approximation of Minima with Improved Asymptotic Speed. Annals of Mathematical statistics 38, 191–200.

[11] Gutjahr, W.J., 2012. Runtime analysis of an evolutionary algorithm for stochastic multi-objective combinatorial optimization. Evolutionary computation 20, 395–421.

[12] Heidrich-Meisner, V., Igel, C., 2009. Hoeffding and bernstein races for selecting policies in evolutionary direct policy search, in: ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, New York, NY, USA. pp. 401–408. doi:http://doi.acm.org/10.1145/1553374.1553426.

[13] Jebalia, M., Auger, A., Hansen, N., 2011. Log-linear convergence and divergence of the scale-invariant (1+1)-es in noisy environments. Algorithmica 59, 425–460. doi:10.1007/s00453-010-9403-3.

[14] Jong, K.A.D., 1992. Are genetic algorithms function optimizers, in: Proceedings of the 2 nd Conference on Parallel Problems Solving from Nature, North. pp. 3–13.

[15] Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., Deb, K., 2002. Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem, in: Parallel Problem Solving from Nature — PPSN VII. Springer Berlin Heidelberg. volume 2439 of *Lecture Notes in Computer Science*, pp. 44–53. doi:10.1007/3-540-45712-7_5.

[16] Leary, S.J., Bhaskar, A., Keane, A.J., 2004. A derivative based surrogate model for approximating and optimizing the output of an expensive computer simulation. Journal of Global Optimization 30, 39–58. doi:http://dx.doi.org/10.1023/B:JOGO.0000049094.73665.7e.

[17] Muhlenbein, H., 1992. How genetic algorithms really work: Mutation and hillclimbing., in: Männer, R., Manderick, B. (Eds.), PPSN, Elsevier. pp. 15–26.

[18] Oliveto, P.S., Witt, C., 2011. Simplified drift analysis for proving lower bounds in evolutionary computation. Algorithmica 59, 369–386. doi:10.1007/s00453-010-9387-z.

[19] Ong, Y.S., Lum, K., Nair, P.B., Shi, D., Zhang, Z., 2003. Global convergence unconstrained and bound constrained surrogate-assisted evolutionary search in aerodynamic shape design, in: Congress on Evolutionary Computation, Special Session on Design Optimisation with Evolutionary Computation(CEC'03), Canberra, Australia. pp. 1856–1863.

[20] Prugel-Benett, A., Rowe, J., Shapiro, J., 2013. Lower bounds on simple ea runtimes in noisy optimization. Personal Communication, Dagstuhl Seminar on the Theory of Evolution Strategies.

[21] Qian, C., Yu, Y., Jin, Y., Zhou, Z.H., 2014. On the effectiveness of sampling for evolutionary optimization in noisy environments, in: Parallel Problem Solving from Nature–PPSN XIII. Springer, pp. 302–311.

[22] Rolet, P., Teytaud, O., 2010a. Adaptive noisy optimization, in: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A., Goh, C.K., Merelo, J., Neri, F., Preuß, M., Togelius, J., Yannakakis,

G. (Eds.), Applications of Evolutionary Computation. Springer Berlin Heidelberg. volume 6024 of *Lecture Notes in Computer Science*, pp. 592–601. doi:10.1007/978-3-642-12239-2_61.

[23] Rolet, P., Teytaud, O., 2010b. Bandit-based estimation of distribution algorithms for noisy optimization: Rigorous runtime analysis, in: Blum, C., Battiti, R. (Eds.), Learning and Intelligent Optimization. Springer Berlin Heidelberg. volume 6073 of *Lecture Notes in Computer Science*, pp. 97–110. doi:10.1007/978-3-642-13800-3_8.

[24] Scharnow, J., Tinnefeld, K., Wegener, I., 2002. Fitness landscapes based on sorting and shortest paths problems, in: Parallel Problem Solving from Nature PPSN VII, Springer Berlin Heidelberg. doi:10.1007/3-540-45712-7_6.

[25] Storch, T., 2006. How randomized search heuristics find maximum cliques in planar graphs, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA. pp. 567–574. doi:10.1145/1143997.1144099.